Denoising Autoencoders In Digital Signal Processing

EE321 Presentation

By - Dibya Darshney

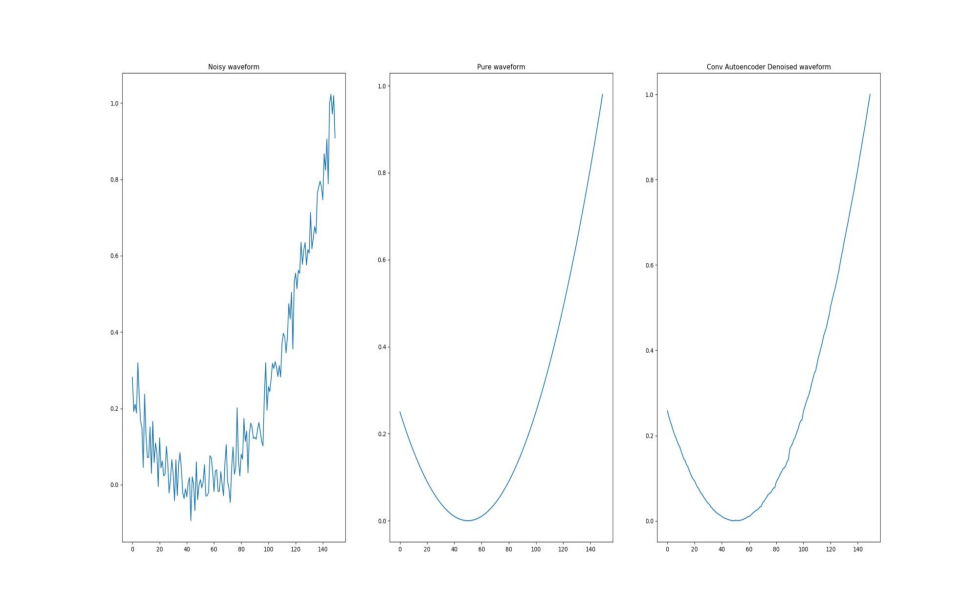
190102025

Abstract

Deep learning-based models have greatly advanced the performance of speech enhancement (SE) systems. Neural networks are used in many tasks today.
One of them is the images processing.

Many algorithms have been proposed for image denoising, including Mean filtering, Gaussian filtering and Bilateral filtering algorithm, each using different local area characteristics to remove noise from image. Methods like the, Wavelet-based denoising method, Total Variate denoising model (TV) method, Sparse Dictionary and Decomposition algorithm, are also widely used in image denoising, and obtain remarkable results. Recently, many researchers try to use deep learning technology for image denoising.

Autoencoder is very popular neural networks for such problems. Denoising autoencoder is an important autoencoder because some tasks we need a preprocessed image to get less noisy result.



Introduction

An **autoencoder** is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”.

The **denoising auto encoder (DAE)** takes a partially corrupted input during training to recover the original undistorted input. Thus noise is intentionally added to the input in order to learn the nonlinear embedding.

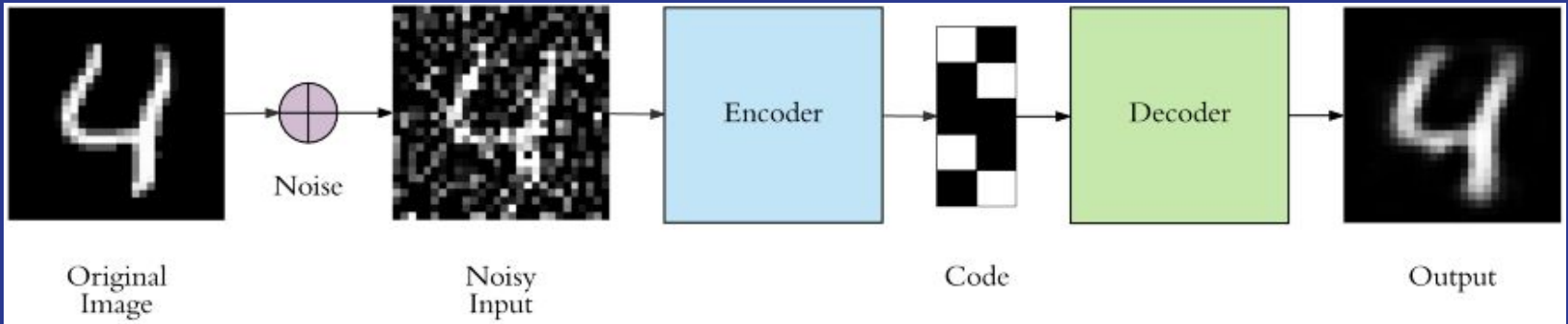
Our goal is to train an autoencoder to perform such pre-processing — we call such models *denoising autoencoders.*

Noise was stochastically (i.e., randomly) added to the input data, and then the autoencoder was trained to recover the original, non perturbed signal.

From an image processing standpoint, we can train an autoencoder to perform automatic image pre-processing for us.

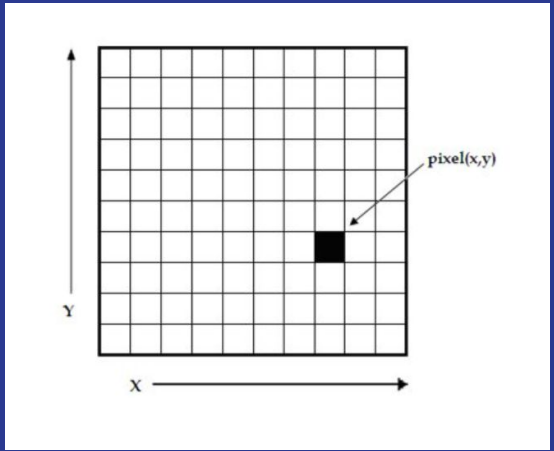
They compress the input into a lower-dimensional code and then reconstruct the output from this representation. The code is a compact “summary” or “compression” of the input, also called the latent-space representation.

An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code. To build an autoencoder we need 3 things: an encoding method, decoding method, and a loss function to compare the output with the target.



Code is a single layer of an ANN with the dimensionality of our choice. The number of nodes in the code layer (code size) is a *hyperparameter* that we set before training the autoencoder.

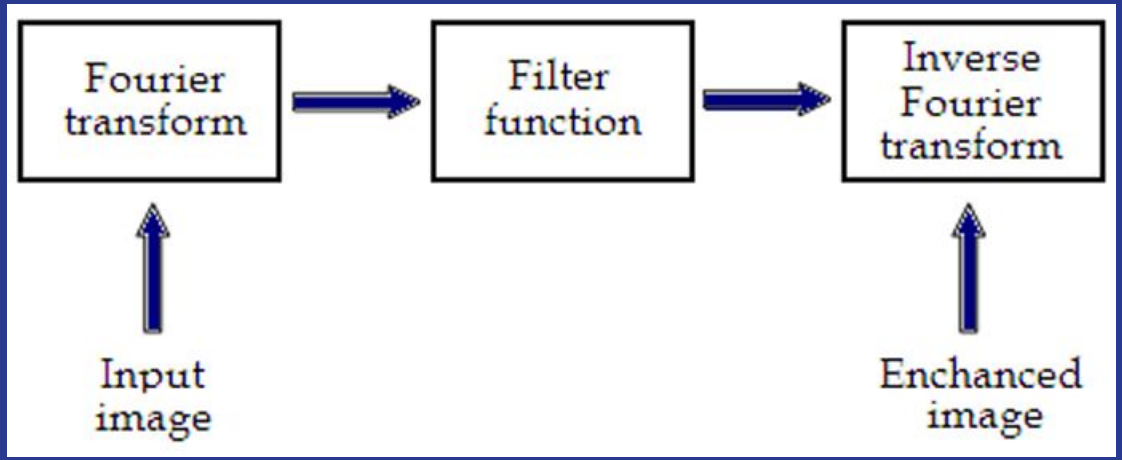
A digital image is a 2D array of pixels. Each pixel is characterised by its (x, y) coordinates and its value.
Digital images are characterised by matrix size, pixel depth and resolution.
The matrix size is determined from the number of the columns (m) and the number of rows (n) of the image matrix (m × n).
The size of a matrix is selected by the operator. Generally, as the matrix dimension increases the resolution is getting better.
Nuclear medicine images matrices are, nowadays, ranged from 64 × 64 to 1024 × 1024 pixels. Pixel or bit depth refers to the number of bits per pixel that represent the colour levels of each pixel in an image.



Digital images

Traditional approaches for performing denoising include methods like

1. Gaussian blur: In Gaussian Blur operation, we take a pixel as the average value of its neighboring pixels. The Gaussian filter is a non-uniform low pass filter that removes the high-frequency components and details.
2. Average blur: During this operation, the image is convolved with a box filter. In this process, the central element of the image is replaced by the average of all the pixels in the kernel area.
3. Median blur: The Median blur operation is resembling the other averaging methods. Here, the middle element of the image is replaced by the median of all the pixels in the kernel dimension. This operation processes the edges while extracting out the noise.
4. Bilateral filter: The Bilateral Filter alters the intensity of each and every pixel with a weighted mean of intensity values from neighboring pixels to obtain an edge-preserving and noise-reducing smoothing effect.



An **image** can be filtered either in the frequency or in the spatial domain. In the first case the initial data is Fourier transformed, multiplied with the appropriate filter and then taking the inverse Fourier transform, re-transformed into the spatial domain.
The filtering in the spatial domain demands a filter mask (it is also referred as kernel or convolution filter).
The filter mask is a matrix of odd usually size which is applied directly on the original data of the image. The mask is centred on each pixel of the initial image.
For each position of the mask the pixel values of the image is multiplied by the corresponding values of the mask. The products of these multiplications are then added and the value of the central pixel of the original image is replaced by the sum.
This must be repeated for every pixel in the image.

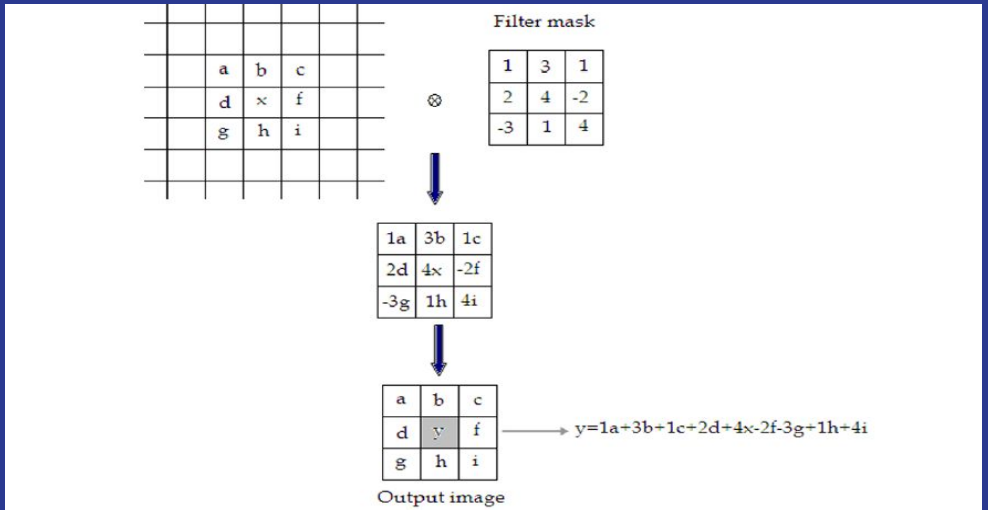


Fig. 4. Illustration of filtering process in spatial domain.

a	b	c
d	e	f
g	h	i

$$\longrightarrow \frac{1}{9}(a+b+c+d+e+f+g+h+i)$$

Fig. 5. Filtering approach of mean filter.

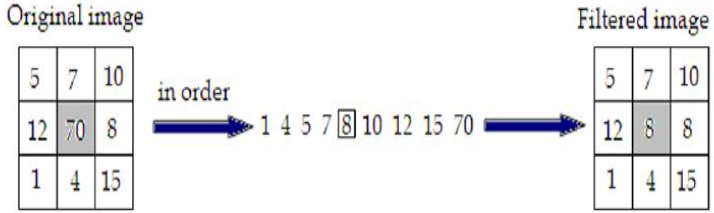


Fig. 6. Filtering approach of Median Filter.

Gaussian filter is a linear low pass filter. A Gaussian filter mask has the form of a bell-shaped curve with a high point in the centre and symmetrically tapering sections to either side (Fig.7). Application of the Gaussian filter produces, for each pixel in the image, a weighted average such that central pixel contributes more significantly to the result than pixels at the mask edges (O'Gorman et al., 2008). The weights are computed according to the Gaussian function (Eq.1):

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} \quad (1)$$

where μ, is the mean and σ, the standard deviation.

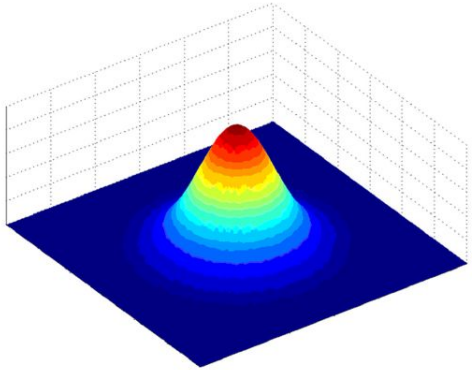


Fig. 7. A 2D Gaussian function.

NO. OF NODES PER LAYER: The autoencoder architecture we're working on is called a *stacked autoencoder* since the layers are stacked one after another. Usually stacked autoencoders look like a "sandwich". The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. As noted above this is not necessary and we have total control over these parameters.

LOSS FUNCTION: We either use *mean squared error (mse)* or *binary cross entropy*. If the input values are in the range [0, 1] then we typically use cross entropy, otherwise we use the mean squared error.

First the input passes through the encoder, which is a fully-connected ANN, to produce the code. The decoder, which has the similar ANN structure, then produces the output only using the code. The goal is to get an output identical with the input. Note that the decoder architecture is the mirror image of the encoder. The targets of the autoencoder are the same as the input.

1. On increasing the number of hyperparameters, it will just mimic the identity function. The autoencoder will reconstruct the training data perfectly, but it will be *overfitting* and won't be able to generalize to new instances.

1. This is why we prefer a "sandwich" architecture, and deliberately keep the code size small. Since the coding layer has a lower dimensionality than the input data, the autoencoder is said to be *under-complete*. It won't be able to directly copy its inputs to the output, and will be forced to learn intelligent features.

2. Keeping the code layer small forced our autoencoder to learn an intelligent representation of the data.

3. There is another way to force the autoencoder to learn useful features, which is adding random noise to its inputs and making it recover the original noise-free data. This way the autoencoder can't simply copy the input to its output because the input also contains random noise. We are asking it to subtract the noise and produce the underlying meaningful data.

We trained the regular autoencoder as follows: `autoencoder.fit(x_train, x_train)`

Denoising autoencoder is trained as: `autoencoder.fit(x_train_noisy, x_train)`

Getting the Noisy Dataset using gaussian noise

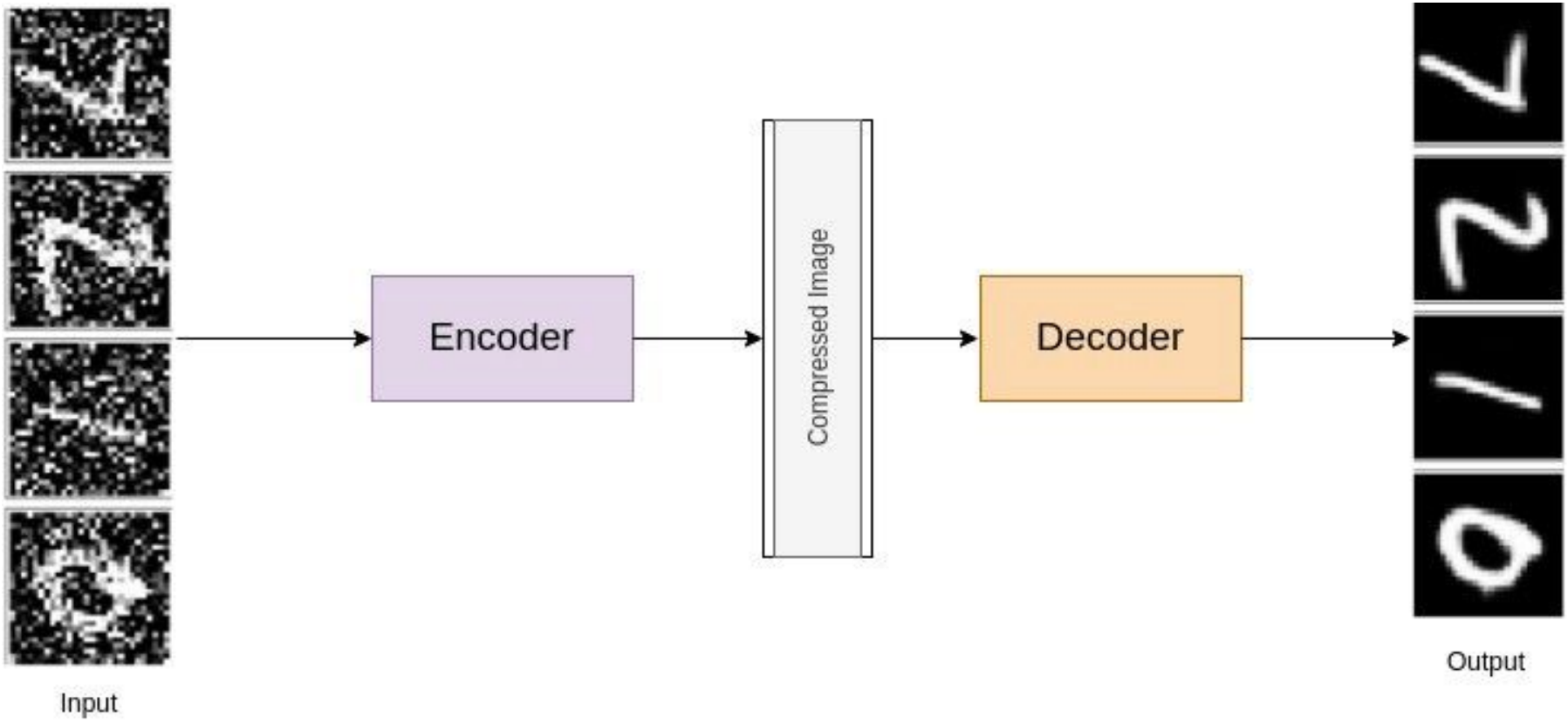
Residual entropy Maximum

According to the principle of information entropy, the degree of ordering of a system is inversely proportional to its information entropy. The information entropy $H(p)$ of a random variable is defined as:

$$H(p) = \sum_{x} p(x) log_2 \left(\frac{1}{p(x)} \right)$$

Under same variance, a Gaussian distribution random variable has the maximum information entropy, which is equal to its variance, that is, a Gaussian distribution random variable contains the least amount of information.

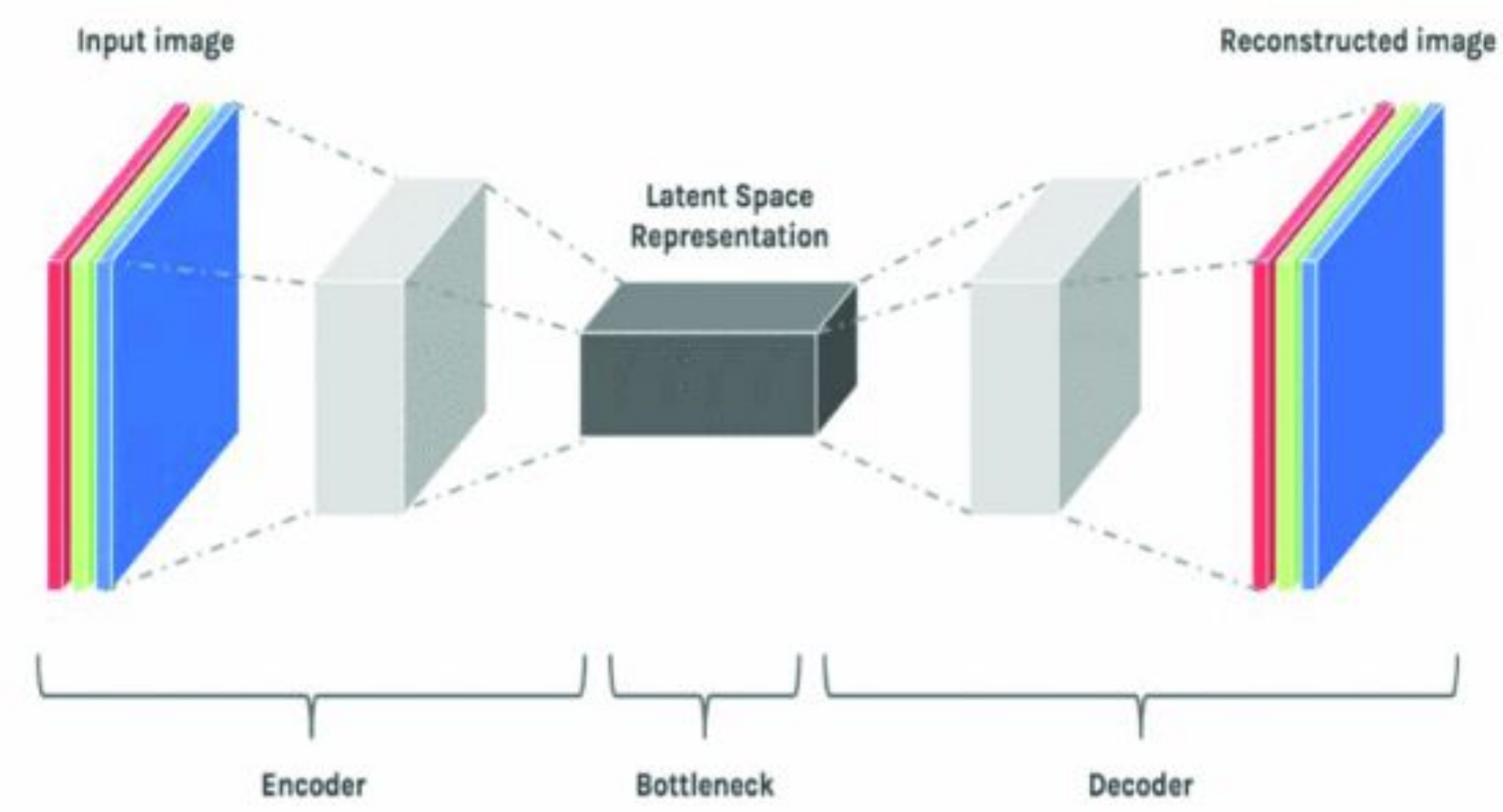
Therefore, to make the residual error= g(f(x))-x contains as little information of as possible, should be as close to a Gaussian distribution white noise as possible.


Encoder
Compressed Image
Decoder
Input
Output

We will use DAE to solve the problem of slow noise reduction. DAE is type of autoencoders that adds noise to inputs during a training process. Autoencoders are one of the unsupervised deep learning models. The aim of an autoencoder is dimensionality reduction and feature discovery. An autoencoder is trained to predict its own input, but to prevent the model from learning the identity mapping, some constraints are applied to the hidden units. The simplest form of an **autoencoder is a feedforward neural network** where the **input x** is fed to the **hidden layer of h(x)** and h(x) is then fed to calculate the **output x** .

$$x = O(a(h)) = \text{Sigmoid}(c + w^* h(x)),$$

$$h(x) = g(a(x)) = \text{Sigmoid}(b + wx)$$


Input image
Reconstructed image
Latent Space Representation
Encoder
Bottleneck
Decoder

Mathematical Representation [4]

R^d, where d is the dimension of data. DAE firstly produces a vector $\tilde{\mathbf{x}}$ by setting some of the elements to zero or adding the Gaussian noise to $\mathbf{x}$. DAE uses $\tilde{\mathbf{x}}$ as input data. The number of units in the input layer is d, which is equal to the dimension of the input data $\tilde{\mathbf{x}}$. The encoding of DAE is obtained by a nonlinear transformation function:

$$\mathbf{y} = f_e(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}), \quad (1)$$

where $\mathbf{y} \in R^h$ denotes the output of the hidden layer and can also be called feature representation or code, h is the number of units in the hidden layer, $\mathbf{W} \in R^{h \times d}$ is the input-to-hidden weights, $\mathbf{b}$ denotes the bias, $\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}$ stands for the input of the hidden layer, and $f_e()$ is called activation function of the hidden layer. We chose ReLU function [17] as the activation function in this study, which is formulated as

$$f_e(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}) = \max(0, \mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}). \quad (2)$$

If the value of $\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}$ is smaller than zero, the output of the hidden layer will be zero. Therefore, ReLU activation function is able to produce a sparse feature representation, which may have better separation capability. Moreover, ReLU can train the neural network for large scale data faster and more effectively than the other activation functions.

The decoding or reconstruction of DAE is obtained by using a mapping function $f_d()$:

$$\mathbf{z} = f_d(\mathbf{W}'\mathbf{y} + \mathbf{b}'), \quad (3)$$

where $\mathbf{z} \in R^d$ is the output of DAE, which is also the reconstruction of original data $\mathbf{x}$. The output layer has the same number of nodes as the input layer. $\mathbf{W}' = \mathbf{W}^T$ is referred

DAE aims to train the network by requiring the output data z to reconstruct the input data x, which is also called reconstruction-oriented training. Therefore, the reconstruction error should be used as the objective function or cost function, which is defined as follows:

$$\text{Cost} = \begin{cases} -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{d}\left[\mathbf{x}_j^{(i)}\log\left(\mathbf{z}_j^{(i)}\right) + \left(1-\mathbf{x}_j^{(i)}\right)\log\left(1-\mathbf{z}_j^{(i)}\right)\right] + \frac{\lambda}{2}\|\mathbf{W}\|^2, & \mathbf{x} \in [0,1], \\ \frac{1}{m}\sum_{i=1}^{m}\left\|\mathbf{x}^{(i)} - \mathbf{z}^{(i)}\right\|^2 + \frac{\lambda}{2}\|\mathbf{W}\|^2, & \text{otherwise,} \end{cases}$$

Noise Learning Based Denoising Autoencoder [3]

Consider a noisy observation Y which consists of the original data X and the noise N, i.e., Y = X + N.

From the information theoretical perspective, DAE attempts to minimize the expected reconstruction error by maximizing a lower bound on mutual information I(X; Y).

What can we do if N is simpler to regenerate than X? It will be more effective to learn N and subtract it from Y instead of learning X directly. In this light, we propose a new denoising framework, named as noise learning based DAE (nlDAE).

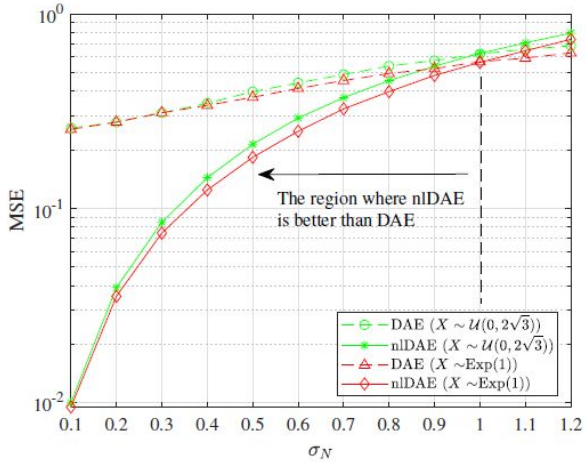


Fig. 2: A simple example of comparison between DAE and nlDAE: reconstruction error according to σ_N.

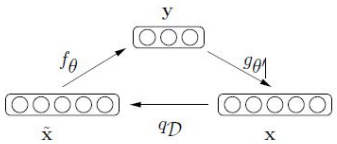
The main advantage of **nlDAE** is that it can maximize the efficiency of the ML approach (e.g., the required dimension of the latent space or size of training dataset) for capability-constrained devices, e.g., IoT, where N is typically easier to regenerate than X owing to their stochastic characteristics.

We first look into the mechanism of DAE to build neural networks. Recall that DAE attempts to regenerate the original data $\mathbf{x}$ from the noisy observation $\mathbf{y}$ via training the neural network. Thus, the parameters of a DAE model can be optimized by minimizing the average reconstruction error in the training phase as follows:

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}\big(\mathbf{x}^{(i)}, g_{\theta'}(f_\theta(\mathbf{y}^{(i)}))\big), \quad (1)$$

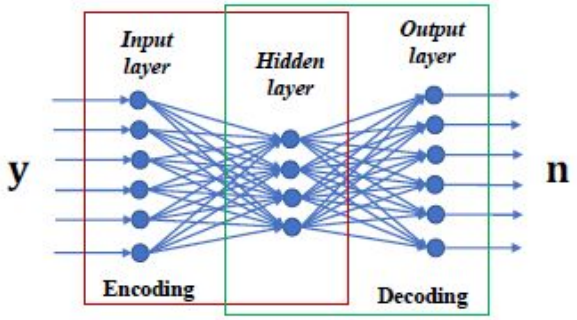
where $\mathcal{L}$ is a loss function such as squared error between two inputs. Then, the j-th regenerated data $\tilde{\mathbf{x}}^{(j)}$ from $\mathbf{y}^{(j)}$ in the test phase can be obtained as follows for all $j \in \{1, \cdots, L\}$:

$$\tilde{\mathbf{x}}^{(j)} = g_{\theta'^*}(f_{\theta^*}(\mathbf{y}^{(j)})). \quad (2)$$

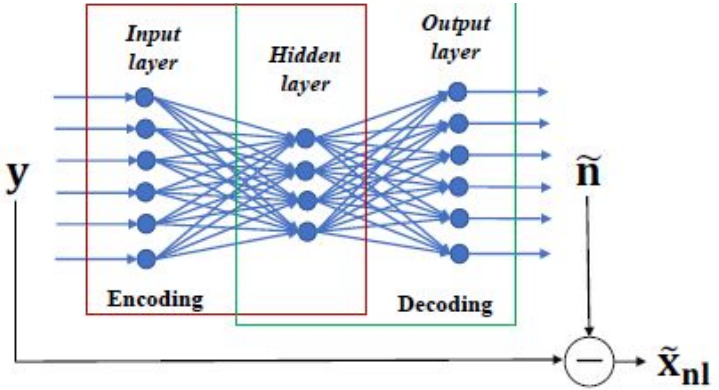


An example $\mathbf{x}$ is corrupted to $\tilde{\mathbf{x}}$. The autoencoder then maps it to $\mathbf{y}$ and attempts to reconstruct $\mathbf{x}$.

- $\mathrm{Ber}, \mathrm{Exp}, \mathcal{U}, \mathcal{N}, \mathcal{CN}$: the Bernoulli, exponential, uniform, normal, and complex normal distributions, respectively.
- $\mathbf{x}, \mathbf{n}, \mathbf{y} \in \mathbb{R}^P$: the realization vectors of random variables X, N, Y, respectively, whose dimensions are P.
- $P' (< P)$: the dimension of the latent space.
- $\mathbf{W} \in \mathbb{R}^{P' \times P}, \mathbf{W}' \in \mathbb{R}^{P \times P'}$: the weight matrices for encoding and decoding, respectively.
- $\mathbf{b} \in \mathbb{R}^{P'}, \mathbf{b}' \in \mathbb{R}^P$: the bias vectors for encoding and decoding, respectively.
- $\mathcal{S}$: the sigmoid function, acting as an activation function for neural networks, i.e., $\mathcal{S}(a) = \frac{1}{1+e^{-a}}$, and $\mathcal{S}(\mathbf{a}) = (\mathcal{S}(\mathbf{a}[1]), \cdots, \mathcal{S}(\mathbf{a}[P]))^T$ where $\mathbf{a} \in \mathbb{R}^P$ is an arbitrary input vector.
- f_θ: the encoding function where the parameter θ is $\{\mathbf{W}, \mathbf{b}\}$, i.e., $f_\theta(\mathbf{y}) = \mathcal{S}(\mathbf{W}\mathbf{y} + \mathbf{b})$.
- $g_{\theta'}$: the decoding function where the parameter θ' is $\{\mathbf{W}', \mathbf{b}'\}$, i.e., $g_{\theta'}(f_\theta(\mathbf{y})) = \mathcal{S}(\mathbf{W}' f_\theta(\mathbf{y}) + \mathbf{b}')$.
- M: the size of training dataset.
- L: the size of test dataset.



(a) Training phase



(b) Test phase

The training and test phases of nlDAE are depicted in Fig. 1. The parameters of nlDAE model can be optimized as follows for all $i \in \{1, \cdots, M\}$:

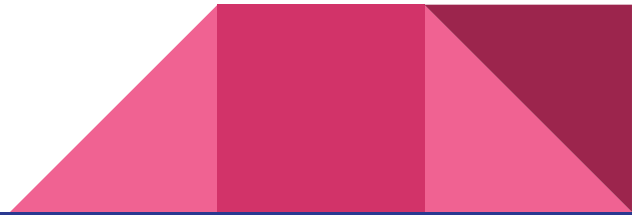
$$\theta_{nl}^{*}, \theta_{nl}^{'*} = \arg\min_{\theta,\theta'} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}\big(\mathbf{n}^{(i)}, g_{\theta'}(f_{\theta}(\mathbf{y}^{(i)}))\big). \quad (3)$$

Notice that the only difference from (1) is that $\mathbf{x}^{(i)}$ is replaced by $\mathbf{n}^{(i)}$. Let $\tilde{\mathbf{x}}_{nl}^{(j)}$ denote the j-th regenerated data based on nlDAE, which can be represented as follows for all $j \in \{1, \cdots, L\}$:

$$\tilde{\mathbf{x}}_{nl}^{(j)} = \mathbf{y}^{(j)} - g_{\theta'^{*}_{nl}}(f_{\theta^{*}_{nl}}(\mathbf{y}^{(j)})). \quad (4)$$

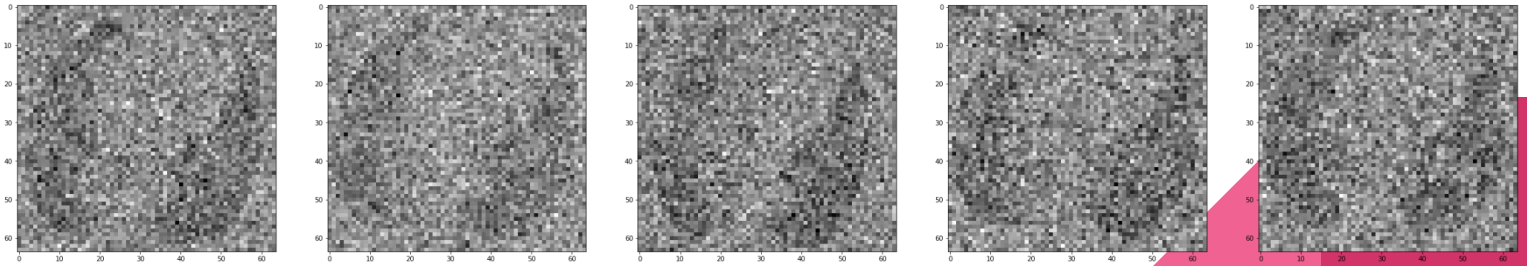
To verify the advantage of nlDAE over the conventional DAE, it has three practical applications : signal restoration, symbol demodulation, and precise localization.

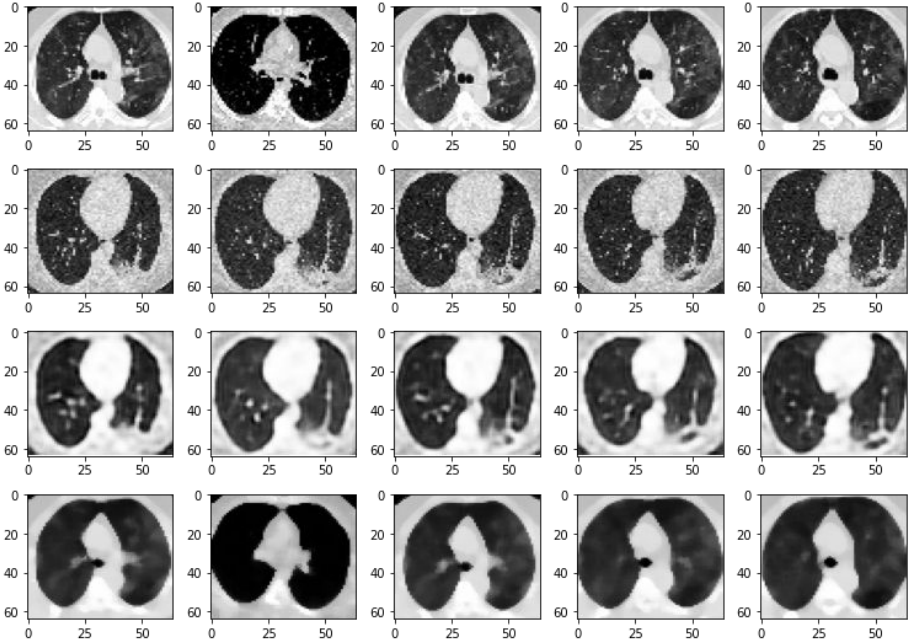
In summary, **nlDAE** outperforms DAE over the whole experiments. nlDAE is observed to be more efficient for the underlying use cases than DAE because it requires smaller latent space and less training data. Furthermore, nlDAE is more robust to the change of the parameters related to the design of the neural network, e.g., the network depth.

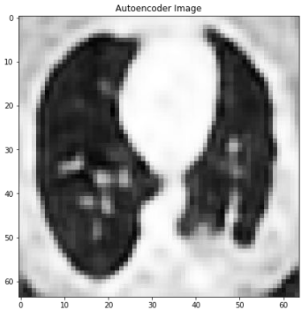


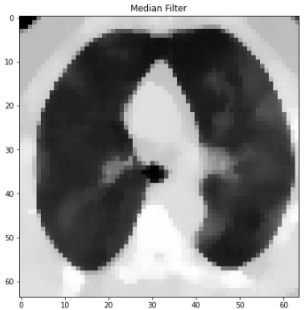
Implementations and Work Done

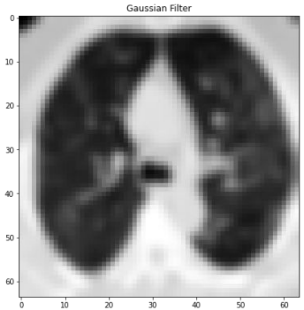
A. Denoising Medical Images
B. Denoising seismic signals

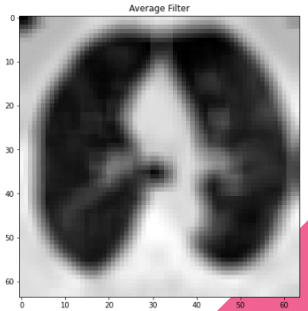


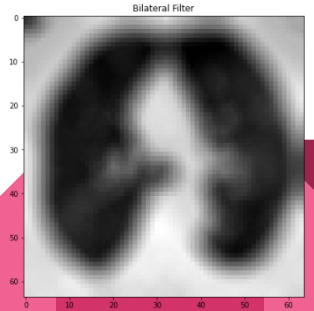



Autoencoder Image


Median Filter


Gaussian Filter


Average Filter


Bilateral Filter

CODE:

```
def autoencoder():
  input_img=Input(shape=(64,64,1),name='image_input')
  #enoder
  x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1')(input_img)
  x = MaxPooling2D((2,2), padding='same', name='pool1')(x)
  x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv2')(x)
  x = MaxPooling2D((2,2), padding='same', name='pool2')(x)

  #decoder
  x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv3')(x)
  x = UpSampling2D((2,2), name='upsample1')(x)
  x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv4')(x)
  x = UpSampling2D((2,2), name='upsample2')(x)
  x = Conv2D(1, (3,3), activation='sigmoid', padding='same', name='Conv5')(x)

  #model
  autoencoder = Model(inputs=input_img, outputs=x)
  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

  return autoencoder
```

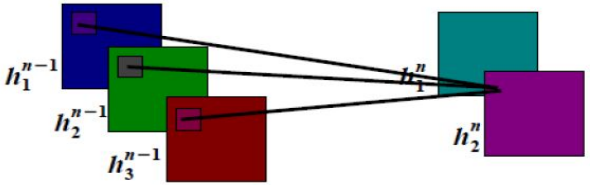
Convolution Layer

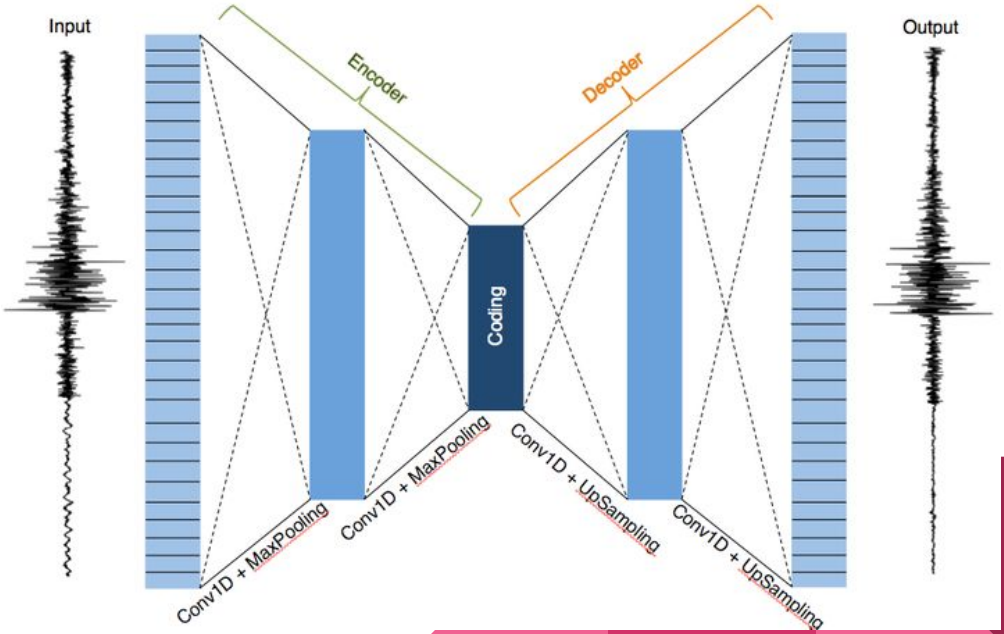
$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel





I’ m using basic CNN architecture to build the model (CNN works well with images). CNN can improve the reconstruction quality. Autoencoders dimensionality reduction (latent space) is quite similar to PCA (Principal Component Analysis) if linear activation functions are used.

Autoencoder consists of two parts:

1. **Encoder:** In Encoder, I am using 2 Conv2D layers and 2 MaxPool2D layers. The output of the 2nd MaxPool2D layer is the encoded features or the input to the Decoder.
2. **Decoder:** Decoder takes the encoder output as input. Here I used Conv2D and UpSampling2D layers. UpSampling2D layer increases the dimension, opposite of MaxPool which reduces the dimension. Output of decoder has same dimension as the input of the encoder.

Code Layer

Autoencoders work by adding a bottleneck in the network. This bottleneck forces the network to create a compressed (encoded) version of the original input. Autoencoders work well if correlations exist between input data and (performs poorly if all the input data is independent).

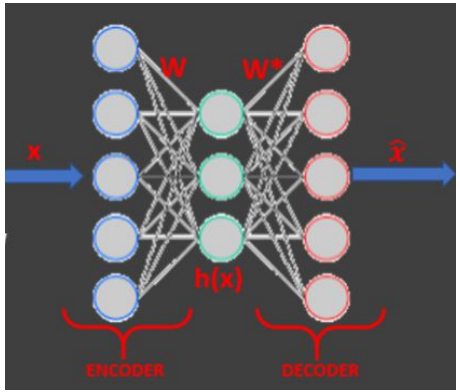
Math behind Autoencoder

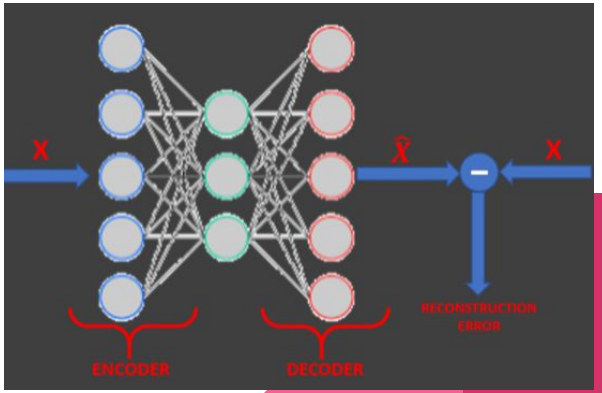
Encoder: `h(x) = sigmoid (W * x + b)`

Decoder: `x^ = sigmoid (W* * h(x) + c`

Reconstruction Error

Autoencoders objective is to minimize the reconstruction error which is the difference between the input X and the network output $\hat{X}$.





Traditionally, autoencoders have been used to learn a feature representation for some data set.

Denoising autoencoders artificially corrupt input data in order to force a more robust representation to be learned.

The most crucial part is the network generation. It is because denoising is a hard problem for the network; hence we'll need to use *deeper* convolutional layers here. It is recommended to start with a depth of 32 for the convolutional layers in the encoder, and the same depth going backwards through the decoder.

The autoencoder's weights are updated by a linear combination of two costs:

the mean squared error between the original image and the reconstructed image.

To measure the quality of the obtained images using different approaches and compare them with our method, we use what is known as the PSNR (Peak signal-to-noise ratio). It is the ratio between the original noiseless image and the error in the image that is first corrupted and then denoised using the particular algorithm. The higher the **PSNR** value, the better the quality of the reconstructed image as it has a higher **Signal Noise ratio**.

Results obtained :

```
PSNR values

Autoencoder Image : 68.59992246352493 dB

Median Filter Image : 56.730625526545936 dB

Gaussian Filter Image : 57.0203347385694 dB

Average Filter Image : 57.173887472252815 dB

Bilateral Filter Image : 57.36302513859329 dB
```

What is PSNR in digital image processing?

Peak signal-to-noise ratio (PSNR) is the ratio between the maximum possible power of an image and the power of corrupting noise that affects the quality of its representation. To estimate the PSNR of an image, it is necessary to compare that image to an ideal clean image with the maximum possible power.
The PSNR block **computes the peak signal-to-noise ratio, in decibels**, between two images. This ratio is used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image.Here, **L** is the number of maximum possible intensity levels (minimum intensity level suppose to be 0) in an image. Where, **O** represents the matrix data of original image. **D** represents the matrix data of degraded image. **m** represents the numbers of rows of pixels and **i** represents the index of that row of the image. **n** represents the number of columns of pixels and **j** represents the index of that column of the image.

RMSE is the root mean squared error.

$$PSNR = 10log_{10}(\frac{(L-1)^2}{MSE}) = 20log_{10}(\frac{L-1}{RMSE})$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(O(i,j) - D(i,j))^2$$

Deep Denoising Autoencoder for Seismic Random Noise Attenuation [6]

Attenuation of seismic random noise is considered as an important processing step to enhance the signal-to-noise ratio (S/N) of the seismic data. A new approach is proposed to attenuate random noise based on a deep denoising autoencoder (DDAE). In this approach, the time-series seismic data are utilized as an input for the DDAE. The DDAE encodes the input seismic data to multiple levels of abstraction, then decodes those levels to reconstruct the seismic signal without noise.

The DDAE is pre-trained in a supervised way using synthetic data, following this the pre-trained model is used to denoise the eld dataset in an unsupervised scheme using a new customized loss function.

Algorithm 1 Proposed algorithm for random noise attenuation.

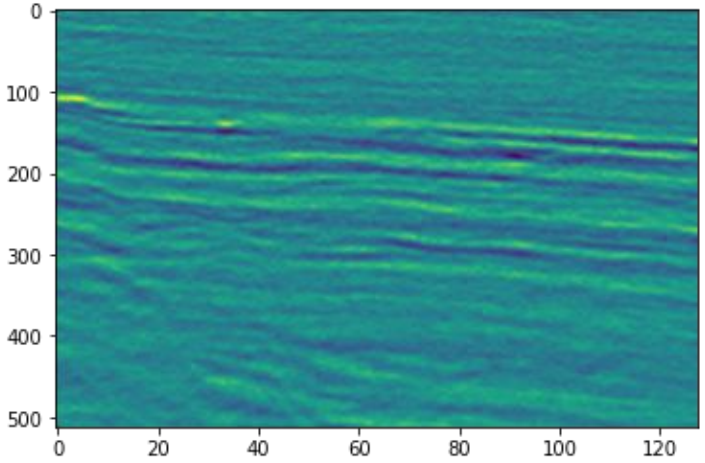
Require: $s(clean\ data), d(noisy\ Data), N = 256$

Ensure: $S\_R(output\ of\ DDAE)$

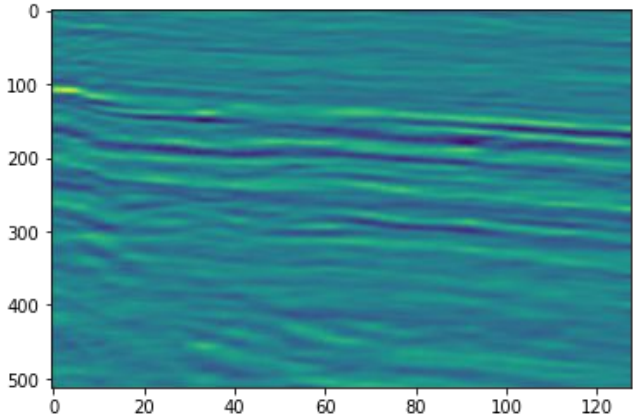
for $epochs = 1$ **to** 1000 **step** 1 **do**
  for $n = 1$ **to** $length(d)$ **step** N **do**
    $d1 \leftarrow d[n : n + 255]$
    $f_1 \leftarrow max(0,\ (W_{e1} \times d1 + b_{e1})\ )$
    $f_2 \leftarrow max(0,\ (W_{e2} \times f_1 + b_{e2})\ )$
    $f_3 \leftarrow max(0,\ (W_{e3} \times f_2 + b_{e3})\ )$
    $f\_r_3 \leftarrow max(0,\ (W_{d1} \times f_3 + b_{d1})\ )$
    $f\_r_2 \leftarrow max(0,\ (W_{d2} \times f\_r_3 + b_{d2})\ )$
    $f\_r_1 \leftarrow max(0,\ (W_{d3} \times f\_r_2 + b_{d3})\ )$
    $S\_R[n : n + 255] \leftarrow W_f \times f\_r_1 + b_f$
    $objective\ function\ l(f) \leftarrow (0.5 \times \sum_n^{n+N-1}(S\_R[n] - s[n])^2)$
    $Update\ \theta = \{W_e, W_d, W_f, b_d, b_e, b_f\}\ using\ Adam\ optimizer$
    return $S\_R[n : n + 255]$
  end for
end for

CODE:

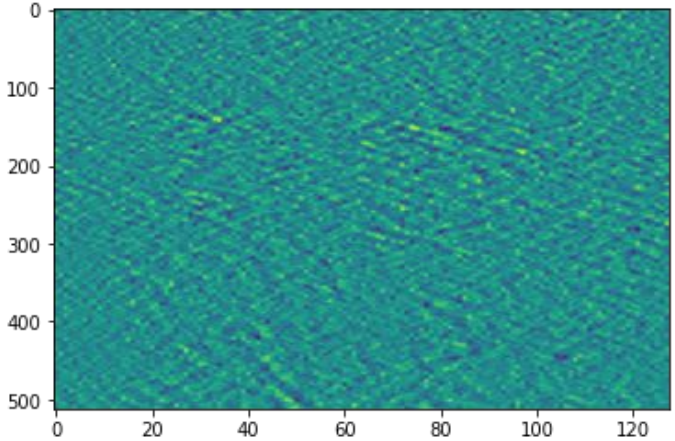
```
mat = scipy.io.loadmat ('./DDAE_SYN.mat')
dataNoise = mat['dn']
dataNoise1= mat['d']
INPUT_SIZE1 = dataNoise.shape[0]
INPUT_SIZE2 = dataNoise.shape[1]

input_img = Input(shape=(INPUT_SIZE2,))

encoded1 = Dense(512, activation='relu')(input_img)
encoded2 = Dense(256, activation='relu')(encoded1)
encoded3 = Dense(128, activation='relu' )(encoded2)

decoded1 = Dense(128, activation='relu' )(encoded3)
decoded2 = Dense(256, activation='relu')(decoded1)
decoded3 = Dense(512, activation='relu' )(decoded2)
decoded = Dense(INPUT_SIZE2, activation='linear')(decoded3)

autoencoder = Model(input_img, decoded)
history = autoencoder.fit(dataNoise,dataNoise1, epochs=50, batch_size=batch, shuffle=True)
```



With Noise



Output after elimination of noise



dataNoise - Output

CONCLUSION and Future work

Using our denoising autoencoder, we were able to remove the noise from the image, recovering the original signal. Denoising autoencoders can be seen as very powerful filters that can be used for automatic pre-processing.

Gaussian distribution should be the best distribution for a image residual generated by denoising method.

To improve generalization ability of our method, the DAE can be trained on dataset which mixed the image samples with different noise levels and types.

Since the speech signal also has structural information, the IDEA is not only limited to image denoising, but also can be used in speech denoising.

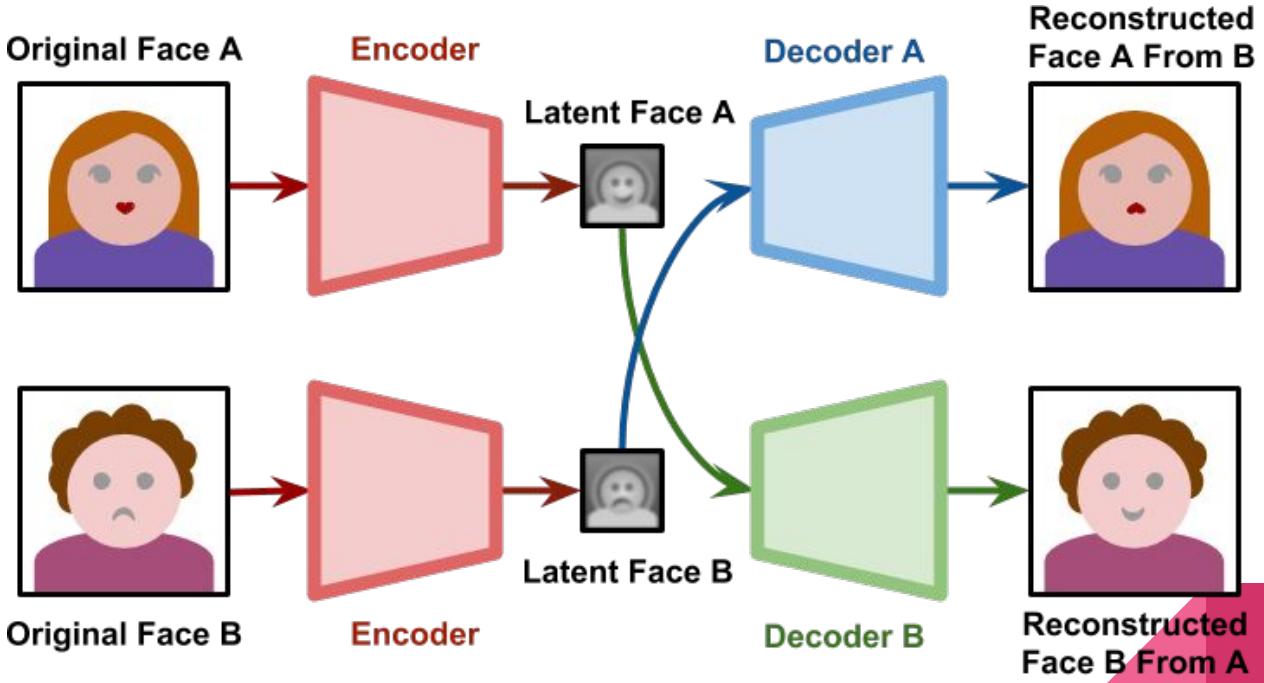
APPLICATIONS

Nowadays, digital images have a valuable role in our daily life, and can be used for various of applications like fingerprint recognition, video surveillance etc. Sometimes, images get infected with noise due to many reasons such as defects in camera sensors, transmission in noisy channel, faulty memory locations in the hardware etc. So, as to improve it for subsequence processing, the noise must be eliminated from the image in advance..

In the context of computer vision, denoising autoencoders can be seen as *very* powerful filters that can be used for automatic pre-processing. For example, a denoising autoencoder could be used to automatically pre-process an image, improving its quality for an OCR algorithm and thereby *increasing* OCR accuracy.

Further Extension/application of Autoencoders

Can be used in the synthesis of deepfake media



References

1. L. Yasenko, Y. Klyatchenko and O. Tarasenko-Klyatchenko, "**Image noise reduction by denoising autoencoder**," 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), 2020, pp. 351-355, doi: 10.1109/DESSERT50317.2020.9125027.
2. https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
3. **Noise Learning Based Denoising Autoencoder** Woong-Hee Lee, Mustafa Ozger, Ursula Challita, and Ki Won Sung, Member, IEEE.
4. **Stacked Denoise Autoencoder** Based Feature Extraction and Classification for Hyperspectral Images
5. Deep Denoising Autoencoder for **Seismic Random Noise Attenuation** Omar M. Saad and Yangkang Cheny
6. **Improved Denoising Auto-encoders for Image Denoising** Qian Xiang Xuliang Pang
7. MATLAB as a Tool in Nuclear Medicine Image Processing, Maria Lyra, Agapi Ploussi and Antonios Georgantzoglou Radiation Physics Unit, A' Radiology Department, University of Athens Greece